unir LA UNIVERSIDAD EN INTERNET

Técnicas de Inteligencia Artificial

Actividad 2. Trabajando con redes neuronales y Deep Learning

Presentado por: Hugo Rodríguez Donato

Fecha: 18/01/2025

Importación de librerias necesarias

Parte I. Regresión

In [1]:
```
# Para esta actividad se importarán las siguientes librerías:
# Pandas, Numpy, Matplotlib, Seaborn, Scikit-learn y tensorflow
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

Cargar el Dataset

Con al menos 1000 instancias, una variable/atributo de la salida, y que dependa de, al menos, 6 variables/atributos de entrada.

In [ ]:
```
# Código para cargar el Dataset

url = 'https://raw.githubusercontent.com/Hugord7/Tecnicas-Inteligencia-Artificial/refs/heads/main/ds_salaries.csv'
```

Descripción de la fuente del Dataset

Haga una descripción de la fuente de dayos utilizada (Incluya los enlaces necesarios).

El dataset contiene los salarios en el ámbito del 'Data Science'. Podemos encontrar diferentes niveles de experiencia, diferentes años, tipos de empleo, residencias y más atributos los cuales vamos a tratar para intentar predecir, según la experiencia, lugar, puesto y más cosas, cual podría ser nuestro salario.

Enlace a Kaggle del dataset: https://www.kaggle.com/datasets/arnabchaki/data-science-salaries-2023?resource=download

Explique el problema a resolver.

Descripción del problema. Tipo de problema (justifique). Variable objetivo, variables de entrada. Utilidad de su posible solución. Elementos adicionales que considere relevantes (no son necesarios contenidos teóricos, sino explicar qué relaciones tratas de comprobar y con qué métodos).

Nos encontramos ante un problema de regresión. Esto significa que queremos predecir una salida numérica. Vamos a tratar de predecir, con las variables de entrada (se detallan a continuación) la variable objetivo, en este caso el salario. Esta solución nos puede ayudar (a nosotros como personas individuales) a calcular el que tendría que ser nuestro salario, dependiendo de las condiciones aportadas, y en cualquier caso nunca nos pagarán menos de lo que realmente valemos. Por otra parte, una empresa podría utilizarlo justamente para lo contrario, predecir sueldos según características del empleado y nunca pagarle más de lo que debería y así ahorrar dinero.

Caracterización del Dataset

Realice una descripción de los datos con:

- Número de instancias en total.

> - Número de atributos de entrada, su significado y tipo.
> - Estadísticas de la variable objetivo.
> - Estadísticas los atributos en relación con la variable objetivo.

Se incorporá una pequeña descripción (EDA) del conjunto de datos utilizado. Se analiza el dataset proporcionando, se muestra al menos algunas de sus características mediante tablas y al menos algunas de ellas en modo gráfico (p.ej., histogramas, diagramas de dispersión, diagramas de cajas y bigotes, etc.)

In [3]:
```
# Código que responde a la descripción anterior
from pandas import read_csv

# Guardamos el dataset en df leyendo el csv sabiendo que esta separado por ','
df = pd.read_csv( url, sep=',', on_bad_lines='skip')

# Imprimimos la cabezera del dataset para hacernos una idea del contenido de este
print('Cabecera Dataset')
print(df.head(5))
print('----------------')

# Con el comando 'shape' podemos ver cuantas instancias y atributos hay en total
print('Instancias y atributos Dataset')
print(df.shape)
print('----------------')

# Con el comando 'describe' analizamos el dataset de una forma general, observando los valores de cada columna
print('Descripción Dataset')
print(df.describe())
print('----------------')
```

```
Cabecera Dataset
   work_year experience_level employment_type                job_title  \
0       2023              SE              FT  Principal Data Scientist
1       2023              MI              CT               ML Engineer
2       2023              MI              CT               ML Engineer
3       2023              SE              FT            Data Scientist
4       2023              SE              FT            Data Scientist

   salary salary_currency  salary_in_usd employee_residence  remote_ratio  \
0   80000             EUR          85847                 ES           100
1   30000             USD          30000                 US           100
2   25500             USD          25500                 US           100
3  175000             USD         175000                 CA           100
4  120000             USD         120000                 CA           100

  company_location company_size
0               ES            L
1               US            S
2               US            S
3               CA            M
4               CA            M
----------------
Instancias y atributos Dataset
(3755, 11)
----------------
Descripción Dataset
         work_year        salary  salary_in_usd  remote_ratio
count  3755.000000  3.755000e+03    3755.000000   3755.000000
mean   2022.373635  1.906956e+05  137570.389880     46.271638
std       0.691448  6.716765e+05   63055.625278     48.589050
min    2020.000000  6.000000e+03    5132.000000      0.000000
25%    2022.000000  1.000000e+05   95000.000000      0.000000
50%    2022.000000  1.380000e+05  135000.000000      0.000000
75%    2023.000000  1.800000e+05  175000.000000    100.000000
max    2023.000000  3.040000e+07  450000.000000    100.000000
----------------
```

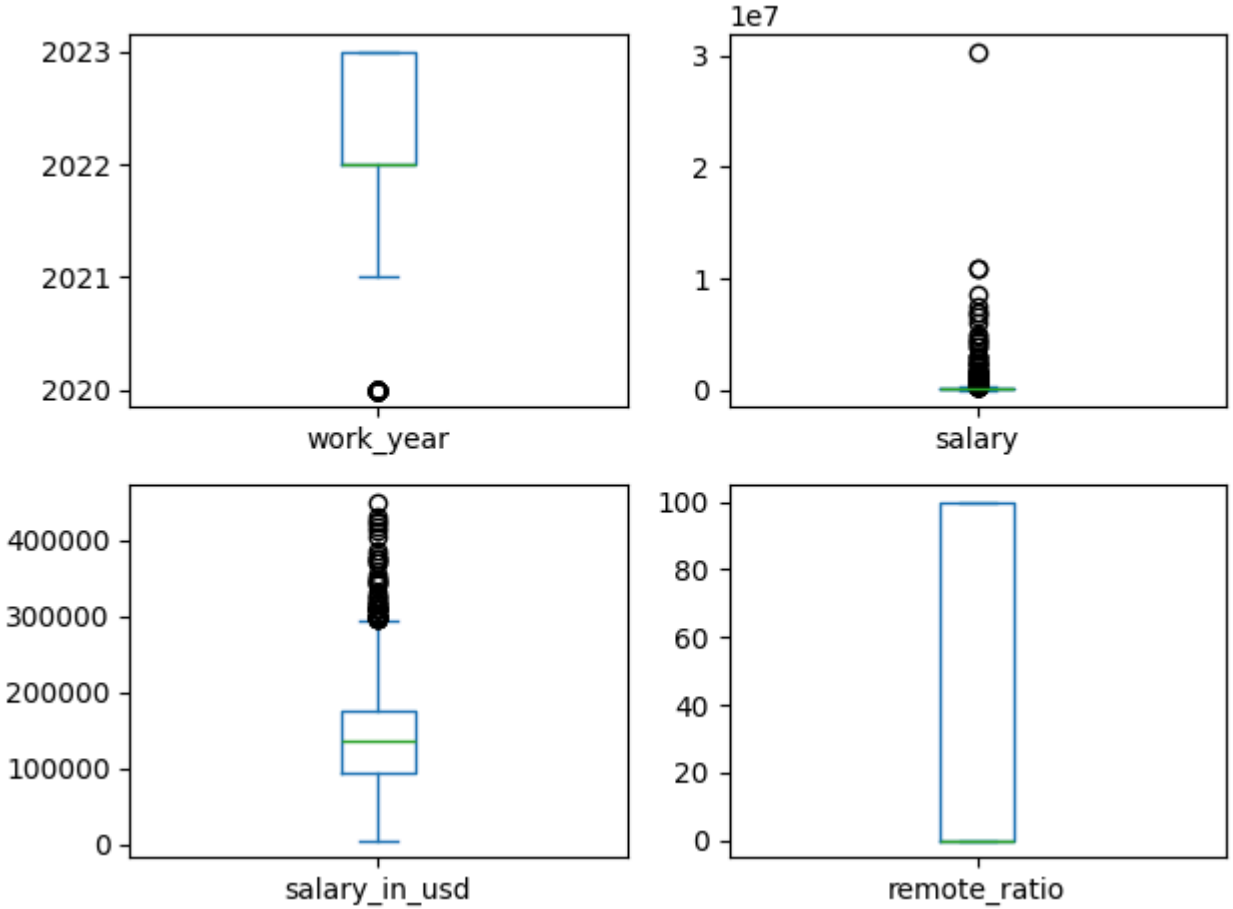
Observamos que la tabla está compuesta por 11 atributos y 3755 instancias.

Antes de describir los atributos de entrada se decide no contabilizar 'salary', 'salary_currency' y 'salary_in_usd' ya que 'salary_in_usd' es nuestra variable objetivo, siendo esta el salario en USD.

Atributos de entrada:

- work_year: Año en el cual se recibió el sueldo (2020, 2021, 2022 y 2023)
- experience_level: Experiencia (EN=Entry-Level, EX=Experienced, MI=Mid-Level y SE=Senior)
- employment_type: Tipo de empleo (CT=Contract Temporal, FL=Freelance, FT=Full-Time)
- job_title: Título de trabajo (ML Engineer, Data Scientist...)
- employee_residence: Residencia del empleado (ES, US...)
- remote_ratio: Tipo de porcentage en remoto (0, 50 y 100)
- company_location: Localización de la compañía (ES, US...)
- company_size: Tamaño de la compañía (S=Small, M=Medium y L=Large)

In [5]:
```
# Código que responde a la descripción anterior (incorpore las lineas de code necesarias. Describa cadas sentencia de código)
from pandas.plotting import scatter_matrix

# Mostramos el diagrama de cajas y bigotes
df.plot(kind='box', subplots=True, layout=(2, 2), sharex=False, sharey=False)
plt.tight_layout()
plt.show()

# # Mostramos el historigrama
df.hist()
plt.tight_layout()
plt.show()
```


2023
2022
2021
2020
work_year
1e7
3
2
1
0
salary
400000
300000
200000
100000
0
salary_in_usd
100
80
60
40
20
0
remote_ratio

En un par de párrafos haga un resumen de los principales hallazagos encontrados:

Como se ha podido observar en la tabla y en las gráficas el salario más repetido está un poco por encima de los 100.000$, siendo la media 137.570$ y el máximo 450.000$. El año que más se podrá evaluar será el 2023, ya que es del que más datos tenemos. Hay pocos trabajos híbridos, recogiéndose la gran mayoría en presencial o remoto.

Preprocesamiento del dataset. Transformaciones previas necesarias para la modelación

In [6]:
```
# Borramos las columnas 'salary' y 'salary_currency' ya que evaluaremos e intentaremos predecir el salario en dolares y estas
# solo hacían la conversión a otras monedas.

df = df.drop(columns=['salary', 'salary_currency'])

# También se comprueba que no hay valores nulos

df.isna().sum()
```

Out[6]:
```
work_year            0
experience_level     0
employment_type      0
job_title            0
salary_in_usd        0
employee_residence   0
remote_ratio         0
company_location     0
company_size         0
dtype: int64
```

In [7]:
```
# Código que realice las transformaciones necesarias para poder realizar los procesos de modelación. Ej.One hot enconding

# Borramos las columnas que se creen innecesarias ('salary' y 'salary_currency') ya que se evaluará e intentará predecir
# sueldo en dolares.
# Filtramos por los valores más habituales en columnas como 'job_title', 'employee_residence' y 'company_location'
df = df[df['job_title'].isin(['Data Engineer', 'Data Scientist', 'Data Analyst', 'Machine Learning Engineer'])]
df = df[df['employee_residence'].isin(['ES', 'CA', 'UK', 'US'])]
df = df[df['company_location'].isin(['ES', 'CA', 'UK', 'US'])]


# Utilizamos este código para comprobar los valores de las columnas
print('----------------')
print(df['experience_level'].unique())
print(df['employment_type'].unique())
print(df['job_title'].unique())
print(df['employee_residence'].unique())
print(df['company_location'].unique())
print(df['company_size'].unique())
print('----------------')
```

```
# Cambiamos los valores de las columnas categóricas por valores númericos
df["experience_level"].replace({"EN": 0, "MI": 1, "SE": 2, "EX": 3}, inplace=True)
df["employment_type"].replace({"CT": 0, "PT": 1, "FL": 2, "FT": 3}, inplace=True)
df["job_title"].replace({"Data Engineer": 0, "Data Scientist": 1, "Data Analyst": 2, "Machine Learning Engineer": 3}, inplace=
df["employee_residence"].replace({"ES": 0, "CA": 1, "UK": 2, "US": 3}, inplace=True)
df["company_location"].replace({"ES": 0, "CA": 1, "UK": 2, "US": 3}, inplace=True)
df["company_size"].replace({"S": 0, "M": 1, "L": 2}, inplace=True)


# Imprimimos los comandos anteriores de nuevo para revisar que se ha ejecutado correctamente, además de confirmar que sigue ha
# de 1000 instancias y 6 atributos

# Imprimimos la cabezera del dataset para hacernos una idea del contenido de este
print('Cabecera Dataset')
print(df.head(5))
print('----------------')

# Con el comando 'shape' podemos ver cuantas instancias y atributos hay en total
print('Instancias y atributos Dataset')
print(df.shape)
print('----------------')
```

```
----------------
['SE' 'MI' 'EX' 'EN']
['FT' 'PT' 'CT' 'FL']
['Data Scientist' 'Data Analyst' 'Machine Learning Engineer'
 'Data Engineer']
['CA' 'US' 'ES']
['CA' 'US' 'ES']
['M' 'L' 'S']
----------------
Cabecera Dataset
   work_year  experience_level  employment_type  job_title  salary_in_usd  \
3       2023                 2                3          1         175000
4       2023                 2                3          1         120000
7       2023                 2                3          1         219000
8       2023                 2                3          1         141000
9       2023                 2                3          1         147100

   employee_residence  remote_ratio  company_location  company_size
3                   1           100                 1             1
4                   1           100                 1             1
7                   1             0                 1             1
8                   1             0                 1             1
9                   3             0                 3             1
----------------
Instancias y atributos Dataset
(2427, 9)
----------------
```

```
C:\Users\hugor\AppData\Local\Temp\ipykernel_14164\4158770743.py:23: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  df["experience_level"].replace({"EN": 0, "MI": 1, "SE": 2, "EX": 3}, inplace=True)
C:\Users\hugor\AppData\Local\Temp\ipykernel_14164\4158770743.py:23: FutureWarning: Downcasting behavior in `replace` is depreca
ted and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To
opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df["experience_level"].replace({"EN": 0, "MI": 1, "SE": 2, "EX": 3}, inplace=True)
C:\Users\hugor\AppData\Local\Temp\ipykernel_14164\4158770743.py:24: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  df["employment_type"].replace({"CT": 0, "PT": 1, "FL": 2, "FT": 3}, inplace=True)
C:\Users\hugor\AppData\Local\Temp\ipykernel_14164\4158770743.py:24: FutureWarning: Downcasting behavior in `replace` is depreca
ted and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To
opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df["employment_type"].replace({"CT": 0, "PT": 1, "FL": 2, "FT": 3}, inplace=True)
C:\Users\hugor\AppData\Local\Temp\ipykernel_14164\4158770743.py:25: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  df["job_title"].replace({"Data Engineer": 0, "Data Scientist": 1, "Data Analyst": 2, "Machine Learning Engineer": 3}, inplace
=True)
C:\Users\hugor\AppData\Local\Temp\ipykernel_14164\4158770743.py:25: FutureWarning: Downcasting behavior in `replace` is depreca
```

```
ted and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To
opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df["job_title"].replace({"Data Engineer": 0, "Data Scientist": 1, "Data Analyst": 2, "Machine Learning Engineer": 3}, inplace
=True)
C:\Users\hugor\AppData\Local\Temp\ipykernel_14164\4158770743.py:26: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  df["employee_residence"].replace({"ES": 0, "CA": 1, "UK": 2, "US": 3}, inplace=True)
C:\Users\hugor\AppData\Local\Temp\ipykernel_14164\4158770743.py:26: FutureWarning: Downcasting behavior in `replace` is depreca
ted and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To
opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df["employee_residence"].replace({"ES": 0, "CA": 1, "UK": 2, "US": 3}, inplace=True)
C:\Users\hugor\AppData\Local\Temp\ipykernel_14164\4158770743.py:27: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.


  df["company_location"].replace({"ES": 0, "CA": 1, "UK": 2, "US": 3}, inplace=True)
C:\Users\hugor\AppData\Local\Temp\ipykernel_14164\4158770743.py:27: FutureWarning: Downcasting behavior in `replace` is depreca
ted and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To
opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df["company_location"].replace({"ES": 0, "CA": 1, "UK": 2, "US": 3}, inplace=True)
C:\Users\hugor\AppData\Local\Temp\ipykernel_14164\4158770743.py:28: FutureWarning: A value is trying to be set on a copy of a D
ataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are set
ting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = d
f[col].method(value) instead, to perform the operation inplace on the original object.
```

```
  df["company_size"].replace({"S": 0, "M": 1, "L": 2}, inplace=True)
C:\Users\hugor\AppData\Local\Temp\ipykernel_14164\4158770743.py:28: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version. To retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df["company_size"].replace({"S": 0, "M": 1, "L": 2}, inplace=True)
```

División del dataset en datos de entrenamiento y datos de test

In [8]:
```
# Código que realice la división en entrenamiento y test, de acuerdo con la estrategia de evaluación planeada. Describa cuál e
# En este apartado se divide el dataset en 80% datos entrenamiento y 20% datos de test. Hay 9 columnas, de las cuales 8 de ell
# y una (salary_in_usd) es atributo de salida (clase)

# Importamos las funciones y métodos a evaluar desde Scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error


from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor

# (División datos para modelos sin red neuronal) Se divide el dataset en 80% datos entrenamiento y 20% datos test ya que
# disponemos de pocos datos para el entrenamiento y evaluación, por esta razón se escoge un número alto de datos de entrenamie

array = df.values
X = np.delete(array, 4, axis=1)  # Eliminar la columna 5 para usarla como entrada
y = array[:, 4]                  # Elegimos la columna 5 para hacer predicciones sobre esta
X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_size=0.20, random_state=1, shuffle=True)

# (División datos para modelos con red neuronal) Se divide el dataset en 80% datos entrenamiento y 20% datos test ya que
# disponemos de pocos datos para el entrenamiento y evaluación, por esta razón se escoge un número alto de datos de entrenamie

train_dataset = df.sample(frac=0.8,random_state=0)
test_dataset = df.drop(train_dataset.index)
```

Propuesta de arquitectura de red neuronal

Describe:

- las neuronas en la capa de entrada
- las capas intermedias – al menos dos –
- capa de salida
- funciones de activación

In [9]:
```
# Quitamos la columna 'salary_in_usd' de los datos 'train_stats'.

train_stats = train_dataset.describe()
train_stats.pop("salary_in_usd")
train_stats = train_stats.transpose()
```

In [10]:
```
# # Quitamos la columna 'salary_in_usd' de los datos de entrenamiento y de los de test

train_labels = train_dataset.pop('salary_in_usd')
test_labels = test_dataset.pop('salary_in_usd')
```

In [11]:
```
# Normalizamos los datos de entrenamiento y los de test

def norm(x):
  return (x - train_stats['mean']) / train_stats['std']
normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)
```

In [12]:
```
# Construimos el modelo

def build_model():
  model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=[len(train_dataset.keys())]),
    layers.Dense(64, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
  ])
```

```
  optimizer = tf.keras.optimizers.RMSprop(0.001)

  model.compile(loss='mse',
                optimizer=optimizer,
                metrics=['mae', 'mse'])
  return model

model = build_model()
```

```
c:\Users\hugor\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [13]:
```
example_batch = normed_train_data[:486]
example_result = model.predict(example_batch)
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
```

In [14]:
```
# Código de la inspección del modelo de red
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	576
dense_1 (Dense)	(None, 64)	4,160
dense_2 (Dense)	(None, 64)	4,160
dense_3 (Dense)	(None, 1)	65

Total params: 8,961 (35.00 KB)

Trainable params: 8,961 (35.00 KB)

Non-trainable params: 0 (0.00 B)

Como se puede observar el modelo consta de 4 capas. Una capa de entrada, dos capas intermedias y una última capa final.

La capa de entrada la forman 64 neuronas. Las dos capas intermedias tienen 64 neuronas cada una. La última capa cuenta con una neurona (la de salida)

La función de activación usada es 'relu', su propósito principal no introducir linealidad en el modelo, lo cual es esencial para aprender relaciones complejas en los datos.

Ajuste de modelo de Regresión RNA

Mediante Python y utilizando al menos Keras sobre TensorFlow 2.0 (tensorflow.keras), entrena el modelo o modelos de red neuronal escogidos.

In [15]:
```
# Código de ajuste y entrenamiento

class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs):
    if epoch % 100 == 0: print('')
    print('.', end='')

EPOCHS = 1000

history = model.fit(
  normed_train_data, train_labels,
  epochs=EPOCHS, validation_split = 0.2, verbose=0,
  callbacks=[PrintDot()])
```

```
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
```

Evaluación de modelo RNA

Defina las estadísticas (métricas) de evaluación, y dividiendo el dataset en datos de entrenamiento, validación y datos de test prueba tu propuesta.

Visualice el progreso de entrenamiento del modelo y muestre las estadísticas de evaluación para los conjuntos de entrenamiento y validación.

In [16]:
```
# Visualización del progreso de entrenamiento
hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
hist.tail()
```

Out[16]:

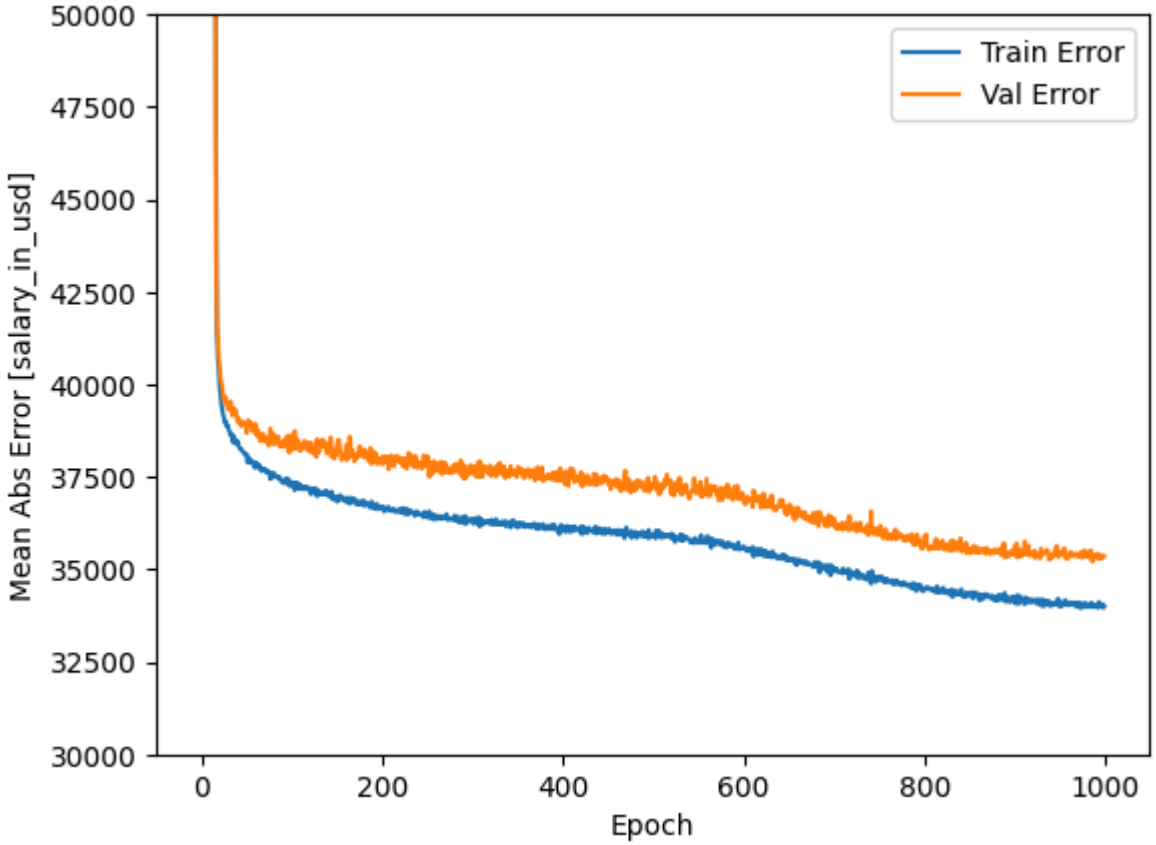
	loss	mae	mse	val_loss	val_mae	val_mse	epoch
995	1.928632e+09	33996.296875	1.928632e+09	2.171924e+09	35294.222656	2.171924e+09	995
996	1.928169e+09	34019.761719	1.928169e+09	2.179354e+09	35375.875000	2.179354e+09	996
997	1.924426e+09	34060.460938	1.924426e+09	2.194105e+09	35309.496094	2.194105e+09	997
998	1.930504e+09	33957.136719	1.930504e+09	2.181177e+09	35324.125000	2.181177e+09	998
999	1.926087e+09	34002.042969	1.926087e+09	2.189957e+09	35357.484375	2.189957e+09	999

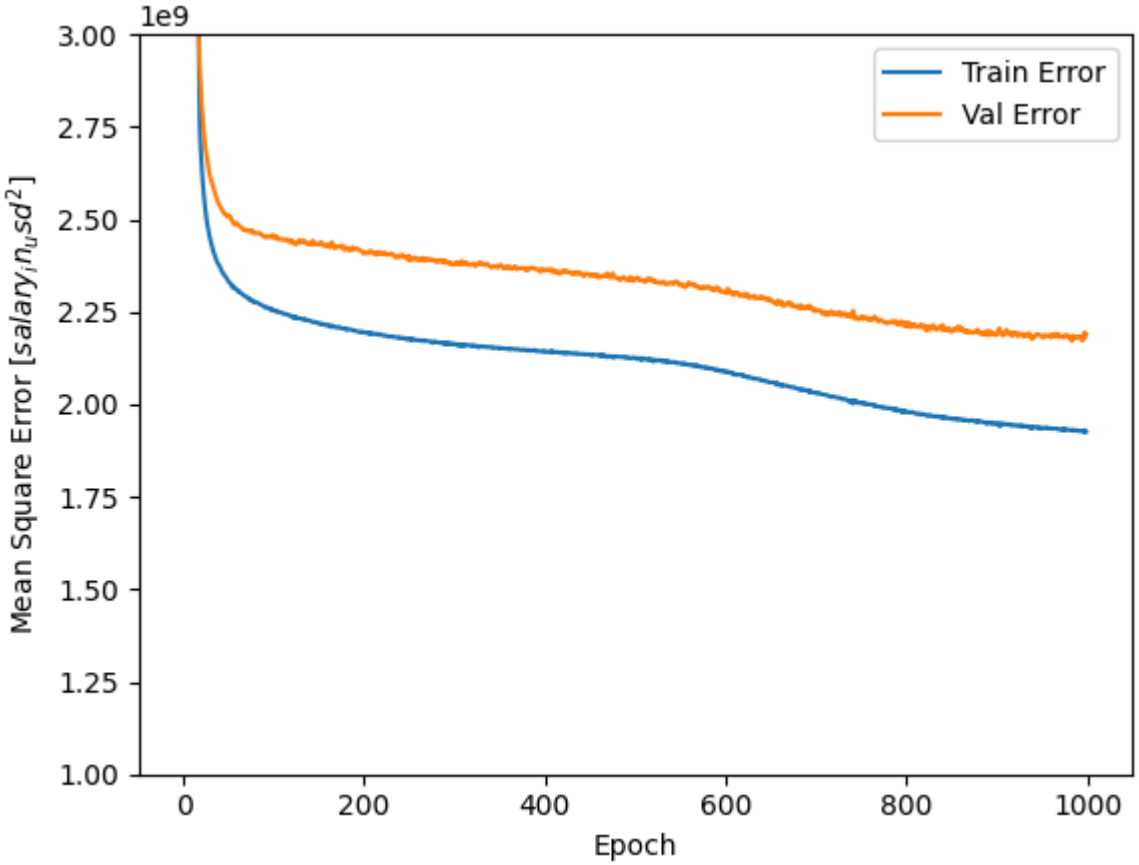
In [17]:
```
# Código de evaluación de la red propuesta (entrenamiento y validación)

nn_loss, nn_mae, nn_mse = model.evaluate(normed_train_data, train_labels, verbose=2)

print("Testing set Mean Abs Error: {:5.2f} salary_in_usd".format(nn_mae))
```

```
61/61 - 0s - 2ms/step - loss: 1974457856.0000 - mae: 34047.9414 - mse: 1974457856.0000
Testing set Mean Abs Error: 34047.94 salary_in_usd
```

In [18]:
```
def plot_history(history):
  hist = pd.DataFrame(history.history)
  hist['epoch'] = history.epoch
```

```
  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Abs Error [salary_in_usd]')
  plt.plot(hist['epoch'], hist['mae'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mae'],
           label = 'Val Error')
  plt.ylim([30000,50000])
  plt.legend()

  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Square Error [$salary_in_usd^2$]')
  plt.plot(hist['epoch'], hist['mse'],
           label='Train Error')
  plt.plot(hist['epoch'], hist['val_mse'],
           label = 'Val Error')
  plt.ylim([1000000000,3000000000])
  plt.legend()
  plt.show()


plot_history(history)
```


Train Error
Val Error
Mean Abs Error [salary_in_usd]
Epoch
50000
47500
45000
42500
40000
37500
35000
32500
30000
0
200
400
600
800
1000



Evalúe los resultados para el conjunto de test.

In [19]:
```
# Código de evaluación de la red propuesta (evaluación conjunto de test)

nn_loss, nn_mae, nn_mse = model.evaluate(normed_test_data, test_labels, verbose=2)

print("Testing set Mean Abs Error: {:5.2f} salary_in_usd".format(nn_mae))
```

```
16/16 - 0s - 6ms/step - loss: 2375972608.0000 - mae: 37643.7617 - mse: 2375972608.0000
Testing set Mean Abs Error: 37643.76 salary_in_usd
```

Ajuste de modelos de Regresión alternativos

Elige al menos un método de regresión no basado en redes neuronales (p.ej. regresión lineal, regresión polinómica, regresión logarítmica, SVR, random forest regression, etc.).

In [20]:
```
# Código de ajuste del modelo 1
model_1 = LinearRegression()
model_1.fit(X_train, Y_train)
predictions_1 = model_1.predict(X_validation)

# Evaluar el modelo
mse = mean_squared_error(Y_validation, predictions_1)
print(f"LinearRegression: Mean Squared Error: {mse}")

# Utilizo validación cruzada estratificada de 10 veces
kfold_1 = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
cv_results_1= cross_val_score(model_1, X_train, Y_train, cv=kfold_1, scoring='neg_mean_squared_error')
print('LinearRegression: Mean Squared Error (Cross-Validation): %f (%f)' % (cv_results_1.mean(), cv_results_1.std()))
```

```
LinearRegression: Mean Squared Error: 2554936964.195798
c:\Users\hugor\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\model_selection\_split.py:776: UserWarning: Th
e least populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(
LinearRegression: Mean Squared Error (Cross-Validation): -2346910286.001124 (221355361.850066)
```

In [21]:
```
# Código de ajuste del modelo 1
model_2 = DecisionTreeRegressor()
model_2.fit(X_train, Y_train)
predictions_2 = model_2.predict(X_validation)

# Evaluar el modelo
mse = mean_squared_error(Y_validation, predictions_2)
print(f"DecisionTreeRegressor: Mean Squared Error: {mse}")

# Utilizo validación cruzada estratificada de 10 veces
kfold_2 = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
cv_results_2= cross_val_score(model_2, X_train, Y_train, cv=kfold_2, scoring='neg_mean_squared_error')
print('DecisionTreeRegressor: Mean Squared Error (Cross-Validation): %f (%f)' % (cv_results_2.mean(), cv_results_2.std()))
```

```
DecisionTreeRegressor: Mean Squared Error: 2237949844.360274
DecisionTreeRegressor: Mean Squared Error (Cross-Validation): -2113435049.241809 (241161345.687500)
```

```
c:\Users\hugor\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\model_selection\_split.py:776: UserWarning: Th
e least populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(
```

In [22]:
```python
# Código para mostrar la evaluación del modelo de regresión LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import root_mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import max_error


print('----------------------')
print('LinearRegression')
print('----------------------')
# Error cuadrático medio
mse = mean_squared_error(Y_validation, predictions_1)
print(f"Mean Squared Error: {mse}")

# Raiz del error cuadrático medio
rmse = root_mean_squared_error(Y_validation, predictions_1)
print(f"Root Mean Squared Error: {rmse}")

# Error absoluto medio
mae = mean_absolute_error(Y_validation, predictions_1)
print(f"Mean Absolute Error: {mae}")

# R² (Coeficiente de Determinación)
r2 = r2_score(Y_validation, predictions_1)
print(f"R2 Score: {r2}")

# Error máximo
max_err = max_error(Y_validation, predictions_1)
print(f"Max Error: {max_err}")

# Validación cruzada
scores = cross_val_score(model_1, X, y, scoring='neg_mean_squared_error', cv=5)
print(f"Cross-Validation MSE: {(-scores.mean())}")
```

```
----------------------
LinearRegression
----------------------
Mean Squared Error: 2554936964.195798
Root Mean Squared Error: 50546.38428409888
Mean Absolute Error: 38967.97248616709
R2 Score: 0.1879929720167529
Max Error: 266616.8702971861
Cross-Validation MSE: 2406599489.421102
```

In [23]:
```python
# Código para mostrar la evaluación del modelo de regresión DecisionTreeRegressor
print('----------------------')
print('DecisionTreeRegressor')
print('----------------------')
# Error cuadrático medio
mse = mean_squared_error(Y_validation, predictions_2)
print(f"Mean Squared Error: {mse}")

# Raiz del error cuadrático medio
rmse = root_mean_squared_error(Y_validation, predictions_2)
print(f"Root Mean Squared Error: {rmse}")

# Error absoluto medio
mae = mean_absolute_error(Y_validation, predictions_2)
print(f"Mean Absolute Error: {mae}")

# R² (Coeficiente de Determinación)
r2 = r2_score(Y_validation, predictions_2)
print(f"R2 Score: {r2}")

# Error máximo
max_err = max_error(Y_validation, predictions_2)
print(f"Max Error: {max_err}")

# Validación cruzada
scores = cross_val_score(model_2, X, y, scoring='neg_mean_squared_error', cv=5)
print(f"Cross-Validation MSE: {(-scores.mean())}")
```

```
----------------------
DecisionTreeRegressor
----------------------
Mean Squared Error: 2237949844.360274
Root Mean Squared Error: 47306.974584729825
Mean Absolute Error: 35647.76316255655
R2 Score: 0.28873744152566405
Max Error: 247000.0
Cross-Validation MSE: 2245455217.8545346
```

Construya un o dos párrafos con los principales hallazgos. Incluye una explicación de los parámetros que consideres relevantes en cada ejecución.

Después de entrenar los modelos sin una red neuronal podemos deducir que no se ha llegado a un buen resultado. Comprobando los resultados en los conjuntos de test, vemos como el error, aún teniendo una cantidad de datos 'pequeña' para predecir un salario dependiendo de muchos factores, es aceptable fijándonos en la media de errores absolutos. En el caso de evaluar R2 podríamos decir que o bien el modelo no es bueno, pero observando los demás resultados se saca como conclusión que faltan datos y atributos para entrenarlo de una manera mas eficiente.

Comparación del desempeño de modelos

Muestra los resultados obtenidos por los diferentes algoritmos escogidos de forma gráfica y comparada/superpuesta.

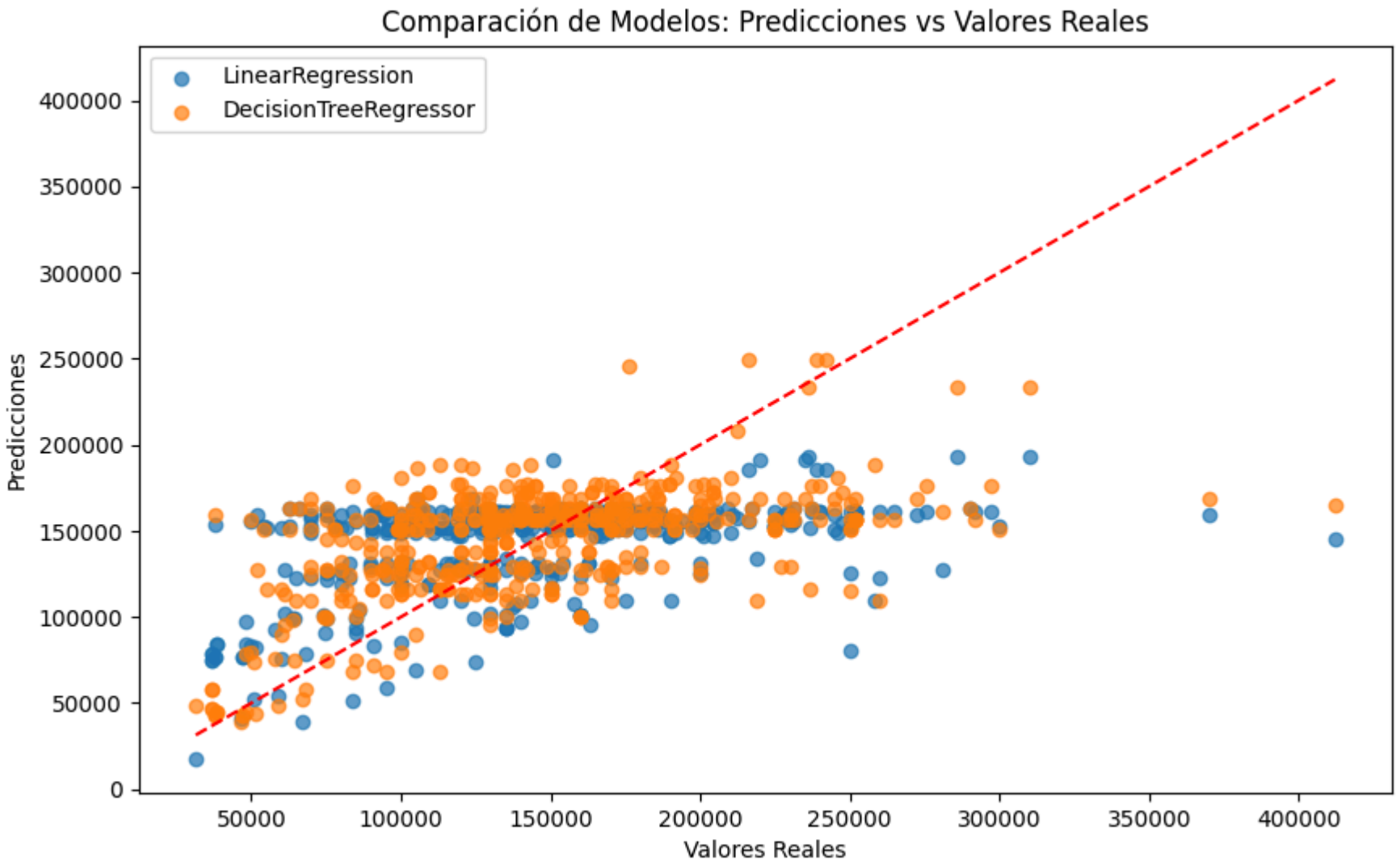
In [24]:
```
# Código para mostrar la comparación de métricas de desempeño de las tres propuestas en tabla
# Creamos un DataFrame con los resultados
metrics = {
    'MSE': [
        mean_squared_error(Y_validation, predictions_1),
        mean_squared_error(Y_validation, predictions_2),
        nn_mse

    ],

    'MAE': [
        mean_absolute_error(Y_validation, predictions_1),
        mean_absolute_error(Y_validation, predictions_2),
        nn_mae
```

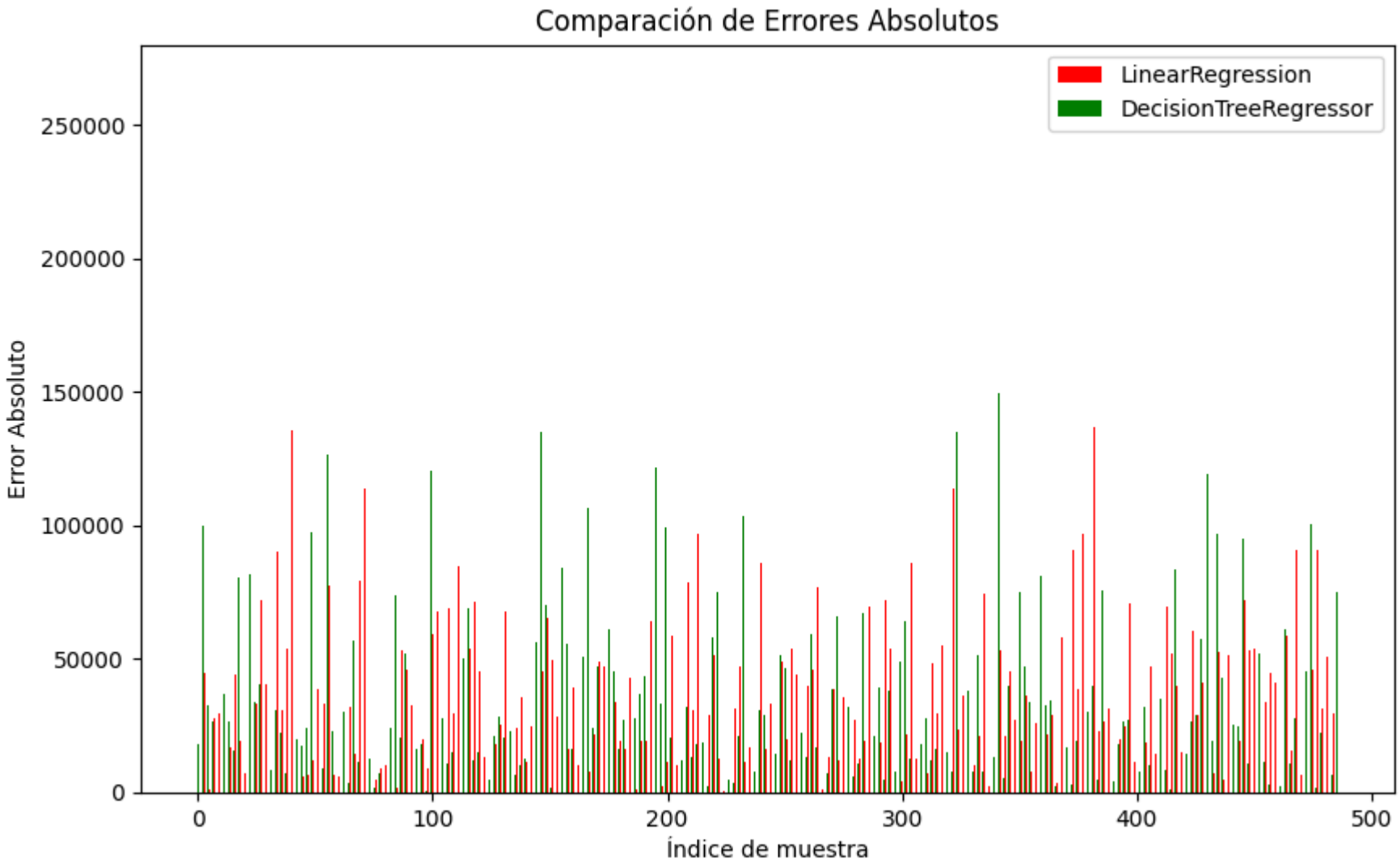
```
    ]
}

results_df = pd.DataFrame(metrics, index=['LinearRegression', 'DecisionTreeRegressor', 'Neural Network'])
print(results_df)
```

```
                            MSE           MAE
LinearRegression       2.554937e+09  38967.972486
DecisionTreeRegressor  2.237950e+09  35647.763163
Neural Network         2.375973e+09  37643.761719
```

In [27]:
```
#Código para mostrar la comparación de métricas de desempeño de las tres propuestas en gráfica

plt.figure(figsize=(10, 6))
plt.scatter(Y_validation, predictions_1, label='LinearRegression', alpha=0.7)
plt.scatter(Y_validation, predictions_2, label='DecisionTreeRegressor', alpha=0.7)
plt.plot([min(Y_validation), max(Y_validation)], [min(Y_validation), max(Y_validation)], color='red', linestyle='--')  # Línea
plt.xlabel('Valores Reales')
plt.ylabel('Predicciones')
plt.title('Comparación de Modelos: Predicciones vs Valores Reales')
plt.legend()
plt.show()




# Calcular errores absolutos
errors_1 = np.abs(Y_validation - predictions_1)
errors_2 = np.abs(Y_validation - predictions_2)

plt.figure(figsize=(10, 6))
index = np.arange(len(Y_validation))
bar_width = 0.25

plt.bar(index, errors_1, width=bar_width, label='LinearRegression', color='r')
plt.bar(index + bar_width, errors_2, width=bar_width, label='DecisionTreeRegressor', color='g')

plt.xlabel('Índice de muestra')
plt.ylabel('Error Absoluto')
plt.title('Comparación de Errores Absolutos')
```

```
plt.legend()
plt.show()
```





Construya un párrafo con los principales hallazgos.

Gracias a la tabla y a las gráficas observamos que, con los datos de entrenamiento el mejor algoritmo ha sido el de redes neuronales, pero a la hora de hacer comprobaciones con los datos de test, el algoritmo con mejor resultado (menor error) ha sido 'DecisionTreeRegressor'. Las predicciones de los salarios como vemos se estancan a partir de 150.000$, lo cual nos indica que los modelos si predicen de manera lógica, pero aún les falta mucho para ser precisos.

Discusión de los resultados obtenidos y argumentos sobre cómo se podrían mejorar de dichos resultados

Realice en este espacio todo el análsis de resultados final incluyendo:

- Resultados comparados. Conclusiones objetivas y significantes con base a las diferentes métricas escogidas.
- Argumentos que describan con qué técnica se obtienen mejores resultados en base a las diferentes métricas que hayas escogido
- Explicación de cómo se podrían mejorar los resultados obtenidos por las redes neuronales, independientemente de que mejoren o no a los algoritmos no basados en redes neuronales.

Conclusiones

Una vez comparados los resultados en el apartado anterior sacamos la conclusión de que, en realidad, ningún algoritmo ha podido resolver el problema que se propuso al principio (predecir el sueldo dependiendo de varios factores). Una vez comentado esto, el mejor algoritmo ha sido el 'DecisionTreeRegressor'. Este algoritmo ha tenido un mejor rendimiento incluso que el algoritmo de la red neuronal, lo que, para mí, supone dos reflexiones. El algoritmo necesita más capas y neuronas para aprender mejor, o se necesita una cantidad muy elevada de datos para resolver este problema. Viendo los resultados obtenidos, me decanto más por la segunda opción, con la cual podemos pensar que, si se dispusiera de más datos para su entrenamiento, los modelos en general harían mejores predicciones.

Comparación

Aclarando porque se piensa que los resultados no han sido buenos, hay que fijarse en las métricas de evaluación:

LinearRegression MSE: 2.554937e+09 MAE: 38967.972486

DecisionTreeRegressor MSE: 2.237748e+09 MAE: 35636.891098

Neural Network MSE: 2.361342e+09 MAE: 37663.578125

Podemos observar cómo los errores son demasiado grandes. Porque aunque los sueldos medios estan por 150.000$, el error medio absoluto roza el 30%. Otra medida que se ha evaluado a conciencia es el R2, el cual nos sugiere que el modelo explica solo el 28.88% de la variabilidad en los datos. Esto nos indica que hay mucho margen de mejora.

Mejoras red neuronal

Mejorar en la predicción del algoritmo de redes neuronales supondría bien, la obtención de una cantidad elevada de datos para poder entrenarlos, o una mejora en la estructura de la red neuronal. Esta mejora podría hacerse ampliando el número de capas o bien las neuronas en estas.